Final Report - CIRTESU

David Eskoundos, Shubh Singhal
Universitat Jaume I
Castellon, Spain

Abstract

Search and rescue missions in underwater environments face challenges like low visibility, dynamic conditions, and precise manipulation. This project developed autonomous algorithms for the BlueROV2 to retrieve a sunken black box in a controlled lab pool, simulating underwater recovery operations. Using YOLO-based object detection, a custom-designed gripper, and PID controllers for depth and yaw stabilization, the system achieved reliable detection, alignment, and grasping of the box handle. The semi-autonomous approach proved robust and generalizable, though challenges with lateral control during approach remain. This work demonstrates the potential of underwater robotics for efficient and reliable recovery tasks.

1 Introduction

Search and rescue missions in underwater environments are inherently challenging, requiring precise navigation, robust object detection, and accurate manipulation in dynamic and often low-visibility conditions. To address these challenges, this project focused on developing and testing autonomous algorithms using the BlueROV2 for the retrieval of a sunken black box in a controlled lab pool. The objective was to simulate a critical component of underwater recovery operations while ensuring reliability and efficiency.

Our system integrates advanced computer vision techniques to detect and localize the black box, even in cluttered underwater settings. A robotic gripper mounted on the BlueROV2 enables secure grasping of the box's handle, ensuring stability during retrieval. Depth sensing and visual feedback, combined with precise motion control algorithms, allow the robot to approach the target with accuracy. Additionally, a PID control system stabilizes the BlueROV2's yaw and depth, enabling seamless movement towards the black box and effective interaction with its environment.

By automating the detection, approach, and retrieval process, this project demonstrates the potential of underwater robotics to enhance safety and efficiency in search and rescue missions. It provides a foundation for more advanced autonomous underwater recovery systems capable of operating in real-world scenarios.

2 Methodology

We use a BlueROV2, equipped with a Newton Sub-Sea Gripper for our project. A custom end-effector for the gripper was also designed and fabricated. For the GCS, we use a laptop with Ubuntu 22 with QGroundControl. We use ROS2 to have mavlink connections between our GCS and ROV for communication over a tethered connection. Every application such as manual control, autonomous control, object detection and grasping has its own ROS2 node.

2.1 Robot Assembly

Physical Setup

As part of this project, we received a BlueROV2 kit from CIRTESU. Our first task was to assemble the BlueROV. We followed the official assembly manual from Blue Robotics [1]. This work includes assembling the overall frame of the ROV, fixing all cable connections of the thruster and other sensors in the main electronics enclosure and finally the battery enclosure. Then, we did a preliminary vacuum test which unfortunately failed. We decided to move along for the time being and installed our thrusters at their respective positions, after finalizing their cable connections to the ESCs. The Figure 1, shows the plan for our thrusters mounting positions. After, we mounted our LED lights in front of the ROV, and finalized its connections with the electronics board. Finally, to make the ROV neutral-buoyant, we added some ballast weights for stability.

Software Setup

The ROV is turned ON and connected to a GCS via a tethered connection. An ethernet connection is created between the two, finally access to BlueOS is established. After some framework updates, we calibrate our thusters using QGroundControl along with our accelerometer and compass. Once everything was setup, we made some final adjustments to the electronics housing to fix our pressure issue. A successful pressure test was done and a first dive was achieved in the CIRTESU tank.

Gripper Design

Initially, the BlueROV2 was equipped with the Newton Sub-Sea Gripper, a standard end-effector designed for general-

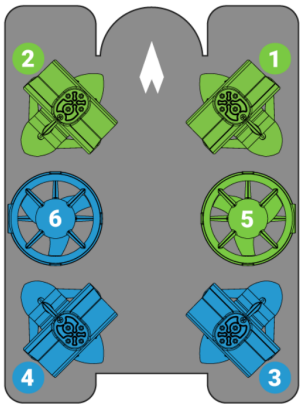

Figure 1: Thruster Configuration of the BlueROV

purpose underwater manipulation tasks. However, to better suit the specific requirements of our mission—retrieving a sunken black box—we decided to remove the original gripper and design a custom end-effector.

Our custom gripper was designed to ensure a secure grasp on the handle of the black box, providing stability during retrieval. The design process involved creating a 3D model of the gripper on Onshape, which can be viewed from here, and then fabricated using 3D printing. The gripper consists of two opposing jaws actuated by a servo mechanism, allowing for precise control over the opening and closing motions.

Figure 2 shows the gripper in its closed position, demonstrating its ability to securely grasp objects. Figure 3 illustrates the gripper in the open position, ready to engage with the target. Figure 4 presents the fully assembled gripper, integrated into the overall system.

The custom design includes several key features, such as a lightweight yet robust structure, enhanced grip strength, and compatibility with the ROV's control system. We addressed challenges such as ensuring watertight seals and optimizing the jaw geometry for reliable grasping.

In conclusion, our custom gripper design offers improved performance for the specific task of retrieving the black box, providing better control and reliability compared to the original Newton Sub-Sea Gripper.

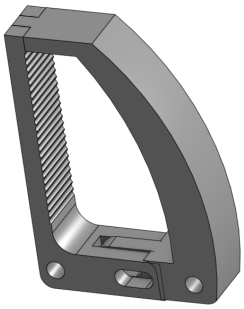

Figure 2: Custom gripper in closed position.

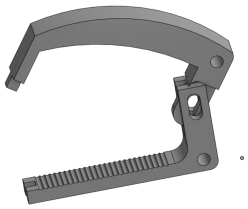

Figure 3: Custom gripper in open position.

Figure 4: Custom gripper fully assembled.

2.2 Control and Stabilization

Manual Controller

To set up autonomous controllers, we needed control access at the low-level for the thrusters. For that we create a ros node - "manual_controller", that creates a mavlink connection between the GCS and ROV over a UDP port. This allows us to send RC input commands to control the thrusters, gripper and the lights. We initialize our joystick as well within this node, and map the joystick commands to the desired robot behavior. The dead zones for the thrusters are also implemented here. We also create 2 different modes - "Manual Mode" where we will only send RC commands from the joystick and "Autonomous Mode" where the controllers will send RC commands and not the user.

PID Controller

We acknowledged that for successful grasping of the black-box, we need to stabilize the ROV's depth and yaw. To this extent, we create separate PID controllers for this in a separate ros node. This node subscribes to the depth data from another ros node which gets the pressure data from the pressure sensor and does the necessary calculations. For yaw, we calculate ROV's yaw with the IMU. In both cases we tune the parameters, and in the end a PD controller works best for both

cases. We used an interactive GUI for tuning the controllers in real-time as shown in Figure 5.

We create 2 modes- "Depth-Hold" and "Yaw-Hold". Normally, there would be a particular depth defined, however when we want to change the depth as such that the user wants the current depth to be the fixed one based on the visual feedback, the Depth-Hold mode is pressed and the program changes the desired depth to the current depth for that instant. Yaw-Hold is similar but fixes the current yaw as the desired yaw.

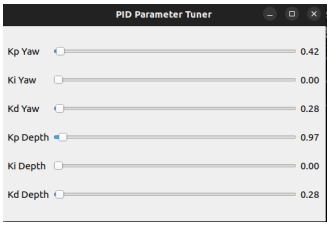

Figure 5: GUI for real-time PID tuning.

2.3 Graphical User Interface (GUI) Design and Implementation

The Graphical User Interface (GUI) is an essential component of our system, providing a user-friendly interface for controlling and monitoring the operations of the BlueROV2. Two implementations of the GUI were developed: one using PyQt5 integrated with ROS2, and the other using Unity integrated with ROS2 through ROS-TCP Connector. Both implementations offer seamless interaction between the operator and the BlueROV2.

Main Features (Common Across Implementations)

- **Camera Feed Display:** A central widget displays real-time video from the ROV's camera, allowing the operator to visually assess the underwater environment.
- **Depth Control Slider:** A vertical slider enables setting the desired depth for the ROV, ranging from 0 to 500, corresponding to 0.0 to 5.0 meters. The selected depth is published to the `/bluerov2/depth_desired` topic.
- **Gain and Alpha Adjustments:** Horizontal sliders allow fine-tuning of gain and alpha values, both ranging from 0 to 20, corresponding to 0.0 to 1.0. These values are published to relevant topics for control algorithms.
- **Mode Toggle:** A toggle switches between manual and automatic modes, with manual mode providing direct control and automatic mode enabling predefined behaviors.
- **Gripper Control Buttons:** Buttons control the gripper's actuation, sending commands to the `/rc_input` topic.
- **Movement Controls:** Buttons for forward, backward, left, and right movements send velocity commands to the `/rc_input` topic.
- **Yaw Hold Toggle:** A toggle enables or disables the yaw hold feature, stabilizing the ROV's orientation.
- **Detection Indicators:** Status indicators show whether the red box and handle are detected, providing immediate feedback.
- **RC Input Display:** A label displays current RC input values, ensuring transparency in control signals.

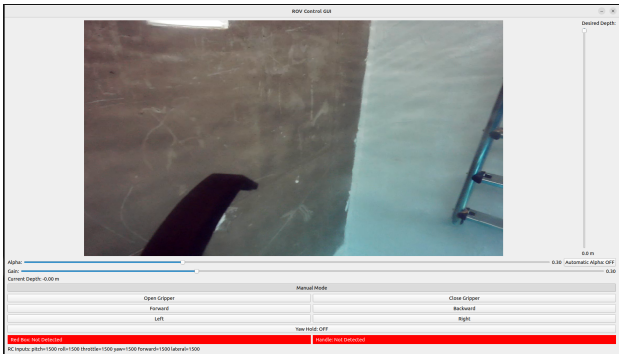

Figure 6: Screenshot of the GUI interface, showcasing the depth control slider, gain and alpha adjustments, and other operational controls.

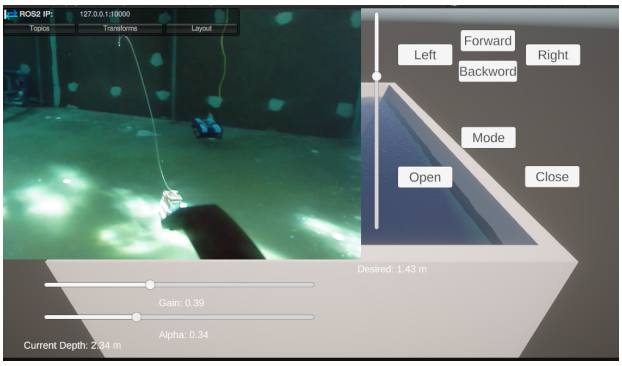

Figure 7: Screenshot of the Unity-based GUI interface, showcasing depth control, gain and alpha adjustments, and other operational controls.

Figures 6 and 7 illustrate the GUI interfaces developed using PyQt5 and Unity, respectively. Each offers unique advantages to enhance the overall user experience.

2.4 Perception and Manipulation

This subsection details the procedures and methodologies employed to enable the BlueROV2 to detect and localize target objects (namely, a red box and its handle) and outlines the subsequent integration of the manipulation tasks.

Perception Pipeline

1. **Data Collection:** Three ROS bag files were recorded under varying lighting conditions and camera angles to capture diverse scenarios.

2. **Image Extraction:** Images were extracted from the recorded bags at intervals of 0.5–1 s, resulting in approximately 476 images.
3. **Dataset Preparation:** The extracted images were uploaded to Roboflow and annotated using the Grounding DINO foundation model. Additional manual annotations were performed for accuracy.

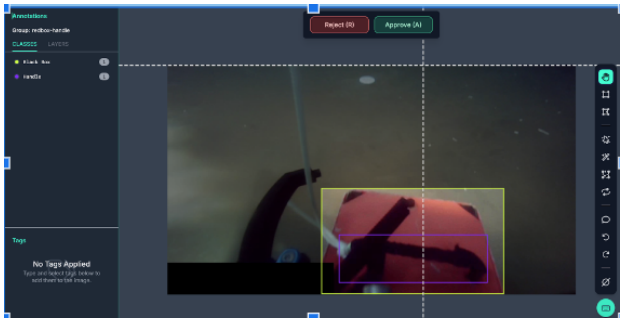

Figure 8: Sample annotation of the dataset, illustrating bounding boxes for both the red box and the handle.

Object Detection Model Training

1. **YOLOv5 Model Training:** Two model configurations were tested:
   - **Small model (95 epochs):** Achieved approximately $P = 91.6\%$ and $R = 90.5\%$.
   - **Medium model (50 epochs):** The training encountered a CUDA out-of-memory error.

     In intermediate experiments, we first attempted to detect only the box, reaching a precision of about 97%. Afterward, the handle class was added to broaden detection capabilities.
2. **Integration:** The final YOLOv5 model (trained weights) was integrated with ROS2 to subscribe to the camera topic and perform real-time object detection on incoming frames.

A short demonstration of the detection process can be viewed online: click here

Center Publishing and Distance Estimation

Once the YOLOv5 model detects the objects, the system publishes the centers of the bounding boxes to dedicated ROS topics (`/redbox_center` and `/handle_center`).

Manipulation

The `RobotGrasper` node is responsible for orchestrating the ROV's manipulation tasks after perception data have been processed. Below is a conceptual overview of how this node works, including the low-pass filtering equations and the mechanism for automatically adjusting the filter gain α.

- **Subscriptions and Data Flow:**
  - It subscribes to the coordinates of the red box and handle (`/redbox_center` and `/handle_center`), allowing real-time updates of each object's image-plane position.

Figure 9: Example detection output from the YOLOv5 model, highlighting both the red box and the handle.

  - It also listens to a distance estimate (`/redbox_distance`) to facilitate adaptive control strategies, as well as an `/alpha` value and a boolean (`/automatic_alpha`) indicating whether to automatically adjust the filter gain.
- **Detection Timeout:**
  - The node stores a timestamp whenever a target (red box or handle) is detected.
  - If a detection message is not received within a specified timeout window (e.g., 0.5 s), the target is considered "lost" or "stale," and the system resets the corresponding object position to `None`.
- **Low-pass Filtering of the Error:**
  - When a valid detection exists, the node computes a horizontal error by comparing the detected x-coordinate of the object to a desired reference (e.g., 600 px).
  - This raw error undergoes normalization (e.g., dividing by the image width). Let $e[k]$ be the new normalized error at time step k, and let $\hat{e}[k]$ be the filtered (low-pass) error. The exponential low-pass filter is given by:

$$\hat{e}[k] \;=\; \alpha\, e[k] \;+\; \big(1-\alpha\big)\,\hat{e}[k-1],$$

    where $0 \le \alpha \le 1$ is the filter gain.
  - The gain α can be set manually (`/alpha` topic) or adjusted automatically based on the estimated distance (`/redbox_distance`). In the automatic case, the node calculates

$$\alpha_{\text{auto}} \;=\; \max\!\big(0,\; \min\!\big(1,\; \frac{d}{2}\big)\big),$$

    where d is the measured distance to the target. This ensures that α remains within the range $[0, 1]$.
- **Publishing Error for Control:**
  - After filtering, the node publishes the final yaw error to `/bluerov2/yaw_error`, which a separate controller can use to adjust the ROV's heading.

Overall, the `RobotGrasper` node serves as an intermediate layer between object detection outputs and ROV motion control. It filters noisy detections, automatically adjusts control gains when desired, and publishes concise yaw error signals for the ROV to respond to, ultimately enabling reliable object approach and grasping maneuvers.

Final Grasping

Once the robot aligns itself horizontally and vertically with respect to the black box, the next step is to manually send short forward pulses to approach the handle. We do it slowly to avoid any sudden fluctuations. Once we are close to the box, the error in yaw becomes bigger due to the relative increase in pixel to cm conversion. To counteract that, we adjust the alpha for our low pass filter to smoothen yaw error. The goal is to make the bottom face of the gripper slide into the handle with just sending forward pulses. Once the gripper attaches, we close it. We then make the robot come to the surface, increasing the motor gains to help with the additional weight.

2.5 Simulation

Alongside analyzing our solutions in the pool, we implemented and tested them in a simulation. For this, we used Blue [2], a ROS2-based pipeline for testing underwater robots. Blue comes with its own model for the BlueROV and allows teleoperation via ROS interfaces. First, we introduced the models for the CIRTESU water tank, black-box, and our custom gripper to the simulation. This was purely for visualization. We implemented our PID controllers for depth and yaw along with the object detection node to see if the robot would detect and correct its position accordingly. The changes we had to make were simply in sending the thruster commands. Unlike the real robot where we used RC Inputs, Blue simulator's teleoperation uses velocity commands over the cmd_vel topic. The GUI was also implemented in the simulation which allowed us to operate and position the ROV. As shown in Figure 10, we can see that that the object detection model working perfectly well, and the visual serving is also sending the required velocity commands to center the frame with the detected object's centroid.

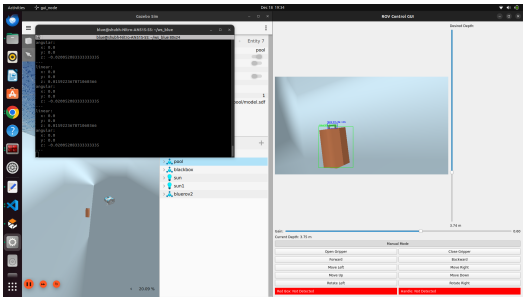

Figure 10: Object detection and Controllers in BLUE Simulator.

3 Results

In terms of control, our depth and yaw controllers achieved great performance in stabilization. We achieved desired yaw control by incoporating a low-pass filter, changing its alpha value manually depending on the distance between the robot and the black-box. We have achieved quite good results with our particular approach of using YOLO based detection. Compared to other approaches like Aruco marker detection, our method is more generalizable as our detection is not limited by the fact that if the marker is in the camera frame or not. The GUI streamlined a lot of tasks for us, ultimately we used a semi-autonomous method because of the feasibility it provided. In terms of depth control, there we not any issues. However, it is true that when we try to approach the target, the lateral shift caused problems. This is the reason we have to try multiple times from different positions to approach and grasp the target. We achieved significant stability while grasping the handle. You can see the video of the full grasping attempt from here.

4 Conclusion

Successful autonomous detection and approach algorithms were developed for the BlueROV. Through a semi-autonomous method, we were able to stabilize the robot and grasp the handle multiple times successfully. The proposed object detection method is much more simpler, and more robust compared to the Aruco marker detection method.

5 Future Work

A number of improvements can be made in this strategy.

- A better gripper: In our case we theorized a gripper with a static base arm. However, we were unable to fabricate it. It should be fairly straightforward and will make the grasping easier.
- Lateral Control: We believe if a suitable lateral controller is implemented, the rate of successful grasping will improve exponentially.

References

[1] Blue Robotics, "Bluerov2 assembly." https://bluerobotics.com/learn/bluerov2-assembly/. Accessed: 2025-01-15.

[2] E. Palmer, C. Holm, and G. Hollinger, "Angler: An Autonomy Framework for Intervention Tasks with Lightweight Underwater Vehicle Manipulator Systems," in *IEEE International Conference on Robotics and Automation*, 2024.